Automating change of representation for proofs in discrete mathematics

Automating change of representation for proofs in discrete mathematics.

Daniel Raggi[1⋆], Alan Bundy[1], Gudmund Grov[2], and Alison Pease[3]

[1] School of Informatics, University of Edinburgh
[2] School of Mathematical & Computer Sciences, Heriot-Watt University
[3] School of Computing, University of Dundee

Abstract. Representation determines how we can reason about a specific problem. Sometimes one representation helps us find a proof more easily than others. Most current automated reasoning tools focus on reasoning within one representation. There is, therefore, a need for the development of better tools to mechanise and automate formal and logically sound changes of representation.
In this paper we look at examples of representational transformations in discrete mathematics, and show how we have used Isabelle's Transfer tool to automate the use of these transformations in proofs. We give a brief overview of a general theory of transformations that we consider appropriate for thinking about the matter, and we explain how it relates to the Transfer package. We show our progress towards developing a general tactic that incorporates the automatic search for representation within the proving process.

Keywords: change of representation, transformation, automated reasoning, Isabelle proof assistant

1 Introduction

Many mathematical proofs involve a change of representation from a domain in which it is difficult to reason about the entities in question to one in which some aspects essential to the proof become evident and the proof falls out naturally.

Many times the transformation makes it explicitly into the written proof, but sometimes it remains hidden as part of the esoteric process of coming up with the proof in the mathematician's mind. For a formal, mechanical proof, this can be problematic, not only because we need to account for the logical validity of the transformation, but because if we want a computational system to find a proof like a mathematician would, we need to be able to incorporate something like the esoteric transformations going on inside the mathematician's mind into the mechanical search.

⋆ This work has been supported by a scholarship from the Mexican Council of Science and Technology (CONACYT).

The importance of representational changes in mathematics is evidenced in historically notable works like Kurt Gödel's incompleteness theorems, where the proof involves matching (or encoding) meta-theoretical concepts like 'sentence' and 'proof' as natural numbers, or more recently Andrew Wiles' proof of Fermat's Last Theorem, which involves matching the Galois representations of elliptic curves with modular forms. This phenomenon is also seen in refinement based formal methods (e.g. VDM and B): one starts with a highly abstract representation that is easy to reason with, and then it is step-wise refined to a very concrete representation that can be implemented as a computer program. All of these transformations are justified by a general notion of morphism.

In this paper we give an overview of a general mathematical framework suitable for reasoning about representational changes in type-theoretic higher-order logics (these are transformations/morphisms between structures that land us in different theories). We see that the operation of Isabelle's *transfer* methods [6] fit into this notion of transformation. It is a way of mechanising inference between two domains, if the system is provided with a transformation by the user. We present a set of transformations we have identified as essential for reasoning in discrete mathematics, and show how we have used the *transfer* tool to implement mechanical proofs in Isabelle that use these transformations. We show our work towards automating the search for representation as a tactic for use within proofs in discrete mathematics in Isabelle.

2 Background

Isabelle/HOL is a theorem proving framework based on a simple type-theoretical higher-order logic [9]. It is one of the most widely used proof assistants for the mechanisation of proofs. Apart from ensuring the correctness of proofs written in its formal language, Isabelle has powerful automatic tactics like `simp` and `auto`, and through time it has been enriched with some internally-verified theorem provers like `metis` [7] and `smt` [11], along with a connection from the internal provers to some very powerful external provers like E, SPASS, Vampire, CVC3 and Z3 through the Sledgehammer tool [10].

The *Transfer* package was first released for Isabelle 2013-1 as a general mechanism for defining quotient types and transferring knowledge from the old 'representation' type into the new 'abstract' type [6]. However, their generalisation is not restricted to the definition of new quotient types, but allows the user to relate any two types by theorems of a specific shape called *transfer rules*. Some of these rules can be defined automatically when the user defines a new quotient type, but the user is free to add them manually, provided that they prove a preservation theorem. Central to this package, the *transfer* and *transfer′* tactics try to automatically match the goal sentence to a new one related by either equivalence or implication, inferring this relation from the transfer rules.

We have taken full advantage of the generality of the transfer package as a means of automating the translation between sentences across domains which are related by what we consider an appropriate and general notion of *structural transformation*. In Section 4 we give an overview of our notion of transformation

and how the tactics of the transfer package are useful mechanisms for exploiting the knowledge of a structural transformation.

3 Overall vision

The worlds of mathematical entities are interconnected. Numbers can be represented as sets, pairs of sets, lists of digits, bags of primes, etc. Some representations are only *foundational* and the reasoner often finds it more useful to discard the representation for practical use (e.g., natural number 3 is represented by $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$ in the typical ZF foundations, but this representation is rarely used in practice), and some are *emergent*; they only come about after a fair amount of accumulated knowledge about the objects themselves, but are more helpful as reasoning tools (e.g., natural numbers as bags of primes). Overall, we think that there is no obvious notion of 'better representation', and it's up to the reasoner to choose, depending on the task at hand. Thus, we envision a system where the representation of entities can be fluidly transformed.

We have looked at problems in discrete mathematics and the transformations commonly used for solving them. Below, we give one motivating example and show how we have mechanised the transformation in question inside Isabelle/HOL. Other motivating examples are briefly mentioned.

Numbers as bags of primes

Let us start with an example of the role of representation in number theory. Consider the following problem:

Problem 1. Let n be a positive integer. Assume that, for every prime p, if p divides n then p^2 also divides n. Prove that n is the product of a square and a cube.

A standard solution to this problem is to take a set of primes p_i such that $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$. Then we notice that the condition "if p divides n then p^2 also divides n" means that $a_i \neq 1$, for each a_i. Then, we need to find $x_1, x_2, \ldots, x_k$ and $y_1, y_2, \ldots, y_k$ where

$$(p_1^{x_1} p_2^{x_2} \cdots p_n^{x_k})^2 (p_1^{y_1} p_2^{y_2} \cdots p_n^{y_k})^3 = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_k}$$

or simply

$$2(x_1, x_2, \ldots, x_k) + 3(y_1, y_2, \ldots, y_k) = (a_1, a_2, \ldots, a_k).$$

Thus, we only need to prove that for every $a_i \neq 1$ there is a pair x_i, y_i such that $2x_i + 3y_i = a_i$. The proof of this is routine.

The kind of reasoning used for this problem is considered standard by mathematicians. However, it is not so simple in current systems for automated theorem proving. The non-standard step is the 'translation' from an expression containing various applications of the exponential function into a simpler form in a linear arithmetic of lists, validated by the fundamental theorem of arithmetic.

The informal nature of the argument, in the usual mathematical presentation, leaves it open whether the reasoning is best thought as happening in an arithmetic of lists where the elements are the exponents of the primes, or perhaps a theory of bags (multisets) where the elements are prime numbers. The reader might find it very easy to fluidly understand how these representations match with each other and how they are really just different aspects of the same thing. Such ease supports our overall argument and vision: that to automate mathematical reasoning, we require a framework in which data structures are linked robustly by logically valid translations, where the translation from one to another is easily conjured up.

The *numbers-as-bags-of-primes* transformation that links each positive integer to the bag of its prime factors is valid because there are operations on each side (numbers and multisets) that correspond to one another. For example, 'divides' corresponds to 'sub(multi)set', 'least common multiple' corresponds to 'union', 'product' corresponds to 'multiset addition', etc. Furthermore, all the predicates used in the statement of problem 1 have correspondences with well-known predicates regarding bags of primes. Thus, the problem can be translated as a whole. Other representations may not be very productive, e.g., try thinking about exponentiation in terms of lists of digits.

Table 1 shows more examples of number theory problems with their corresponding problem about multisets.

Problem in $\mathbb{N}$	**Problem in multisets**
Prove that there is a unique set $\{x, y, z\}$ with different x, y, z greater than 1, such that $xyz = 100$.	Prove that there is a unique way to partition $\{2, 2, 5, 5\}$ into three different non-empty parts.
Prove that in a set of 9 natural numbers, where none is divided by a prime larger than 6, there is a pair whose product is a perfect square.	Take 9 multisets whose only elements are 2, 3 and 5. Prove that two of the multisets have multiplicities with the same parity.

Table 1. Number theory problems and their multiset counterparts.

Numbers as sets

Many numerical problems have *combinatorial proofs*. Theses are proofs where numbers are interpreted to be cardinalities of sets, and the whole problem can be converted to a problem about sets.

Enumerative combinatorics studies how sets relate to their cardinalities. As such, its theorems provide the link that allows us to translate numerical problems into finite set-theoretical problems.

Table 2 shows examples of arithmetic problems with their corresponding finite set theory problems. While the proofs of the numerical versions are not obvious at all (some of which are important results in basic combinatorics), the proofs of their finite set versions can be considered routine.

Problem in $\mathbb{N}$	**Problem in sets**
$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$	The set $\{x \subseteq \{0,1,\ldots,n\} : \lvert x\rvert = k+1\}$ can be partitioned into 2 parts: those that contain element n and those that don't.
$\frac{n(n+1)}{2} = 1 + 2 + \cdots + n$	The set $\{x \subseteq \{0,1,\ldots,n\} : \lvert x\rvert = 2\}$ can be partitioned into n parts $X_1, X_2, \ldots, X_n$ where the largest element of each $x \in X_i$ is i.
$2^{n+1} - 1 = \sum_{i=0}^{n} 2^i$	The power set of $\{0,1,\ldots,n\}$, excluding the empty set, can be partitioned into n parts $X_1, X_2, \ldots, X_n$ where the largest element of each $x \in X_i$ is i.
$2^n = \sum_{i=0}^{n} \binom{n}{i}$	The power set of $\{1,\ldots,n\}$ can be partitioned into $n+1$ parts $X_0, X_1, \ldots, X_n$ where $\lvert x\rvert = i$ for every $x \in X_i$.

Table 2. Numerical problems and their set counterparts.

Interconnectedness

We want to stress the importance of having fluidity of representations. For example, we talked about the ease with which we could think that the *numbers-as-bags-of-primes* transformation is actually a transformation of numbers to a theory of lists, where elements of the list are the exponents of the ordered prime factors. Inspired by this, we have mechanised many other simple transformations, but whose composition allows us to translate fluently from one representation to another. Our global vision of transformations useful in discrete mathematics, which we have mechanised[4], is represented in Figure 1. It is worth mentioning that the diagram is not commutative and that it abstracts logical relations (information may be lost, so some paths can only be traversed in one direction).

In the next section we show how a notion of transformation that accounts for this kind of correspondence between structures can be applied in formal proofs using Isabelle's Transfer tool.

4 On Transformations and the Transfer tool

In this section we give a brief overview of a very general theory of transformations. We do not claim originality of the essence of this theory. However, we believe that the presentation we give brings clarity to the problem. We explain how Isabelle's Transfer tool relates to it. Consider the following definitions:

Definition 1. *A **domain** is a class of entities and a set of types, where each entity of the domain corresponds to exactly one type.*

[4] These can be found in http://homepages.inf.ed.ac.uk/s1052074/AutoTransfer/. They are updated regularly.

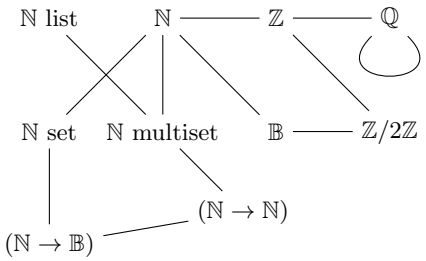

Fig. 1. Nodes stand for theories connected by transformations useful in discrete mathematics. Apart from the aforementioned transformations, it includes other simpler ones. Actually, some of these transformations, such as that connecting ℕ list and ℕ set, are polymorphic, but presented in the diagram as relating only to type ℕ. This is because the numbers-as-bags-of-primes transformation is not polymorphic.

Definition 2. *A **transformation** from a domain $\mathscr{X}$ to a domain $\mathscr{Y}$ is a collection $\mathscr{R}$ where every $R \in \mathscr{R}$ is a relation $R : X \to Y \to \mathbb{B}$ between a type X of domain $\mathscr{X}$ and a type Y of domain $\mathscr{Y}$.*[5]

This relational notion of a transformation makes it possible to account for partial and multivalued mappings in a logic like Isabelle's HOL.

We consider a *structure* to be the class conformed by all the entities of the closure of a domain under a set of type constructors. In this work, we focus on structures containing type $\mathbb{B}$, generated with the *function type constructor* $\to$, because the basis of a higher order logic can be fully expressed under such a structure. Then, if our domain has entities of types A and B, its structure under $\to$ has all the entities of types $A \to B$, $A \to B \to A$, etc.

Preservation of structure is captured with the use of structural *relators*, which can be thought of as rules for extending relations (transformations) to the structures of their domains. In particular, given that our work is based on Isabelle/HOL and on the Transfer package, we focus on one relator.

Definition 3. *The **standard functional extension** of two relations $R_A : A \to A' \to \mathbb{B}$ and $R_B : B \to B' \to \mathbb{B}$ (written $R_A \Rightarrow R_B$) is a relation that relates two functions $f : A \to B$ and $f' : A' \to B'$ whenever they satisfy the following property:*

$$\forall\, a : A.\ \forall\, a' : A'.\ \ [R_A\, a\, a' \longrightarrow R_B\, (f\, a)\, (f'\, a')]$$

We call the operator $\Rightarrow$ the **standard function relator**. Intuitively, $(R_A \Rightarrow R_B)\, f\, g$ means that f and g send arguments related by R_A to values related by R_B. This relator allows us to express how functions (and by extension relations) map to each other in a way that the structure of the domain is preserved.

For the numbers-as-bags-of-primes transformation, consider relation $\mathcal{F} : \mathbb{N} \to \mathbb{N}$ `multiset` $\to \mathbb{B}$, that relates every positive integer with the multiset of its prime factors.

[5] $\mathbb{B}$ stands for type of booleans.

Example 1. Let $* : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ be the usual multiplication and $\uplus : \mathbb{N}$ `multiset` $\to \mathbb{N}$ `multiset` $\to \mathbb{N}$ `multiset` the 'addition' of multisets (in which the multiplicities are added per element). Then we have $(\mathcal{F} \Rightarrow (\mathcal{F} \Rightarrow \mathcal{F})) * \uplus$ (also written $(\mathcal{F} \Rightarrow \mathcal{F} \Rightarrow \mathcal{F}) * \uplus$).
Note that, by expanding the definition of $\Rightarrow$ in $(\mathcal{F} \Rightarrow (\mathcal{F} \Rightarrow \mathcal{F})) * \uplus$ we get

$$\forall\, n_1.\, \forall\, N_1.\, \mathcal{F}\, n_1\, N_1 \longrightarrow (\forall\, n_2.\, \forall\, N_2.\, \mathcal{F}\, n_2\, N_2 \longrightarrow \mathcal{F}\, (n_1 * n_2)\, (N_1 \uplus N_2))$$

which is equivalent to

$$\forall\, n_1, n_2.\, \forall\, N_1, N_2.\ (\mathcal{F}\, n_1\, N_1 \wedge \mathcal{F}\, n_2\, N_2) \longrightarrow \mathcal{F}\, (n_1 * n_2)\, (N_1 \uplus N_2)$$

This demonstrates how nesting the operator $\Rightarrow$ preserves its intuitive definition: 'related arguments map to related values'. In this particular case, this is true due to the law of exponents $p^a p^b = p^{a+b}$.

Furthermore, the matching of relations can also be expressed with the help of $\Rightarrow$, using a boolean relation, as demonstrated by the example below with equivalence (boolean equality) `eq` : $\mathbb{B} \to \mathbb{B} \to \mathbb{B}$.

Example 2. Let `div` : $\mathbb{N} \to \mathbb{N} \to \mathbb{B}$ be the relation such that `div` $n\, m$ whenever n divides m (also written $n|m$), and $\subseteq$: $\mathbb{N}$ `multiset` $\to \mathbb{N}$ `multiset` $\to \mathbb{B}$ the relation such that $a \subseteq b$ whenever the multiplicity of each element of a is lesser or equal to its multiplicity in b. Then, we have $(\mathcal{F} \Rightarrow \mathcal{F} \Rightarrow$ `eq`$)$ `div` $\subseteq$, because n divides m if and only if every prime is contained at least as many times in the multiset-factorisation of m as it is in n.

Logical matches (preservation of truth values) can also be expressed across structures, e.g., (`eq` $\Rightarrow$ `eq` $\Rightarrow$ `eq`) `imp` `imp` represents that implication `imp` preserves truth if its arguments are replaced by equivalent ones. Other interesting logical matches can be expressed as well.

The general notion of transformation above tells us how theories will relate to one another. Isabelle's Transfer method is an algorithm for transforming a sentence using knowledge about one of these transformations. The simple standard function relator is at the basis of the method. We give a short introduction next.

4.1 Transforming sentences with the Transfer tool

When trying to prove a sentence β we want to find another sentence α such that $\alpha \longrightarrow \beta$, along with a proof for α. In particular, if β talks about a domain B and we know a structural transformation from a domain A to B, we might be able to find an α about A, such that $\alpha \longrightarrow \beta$.

Isabelle's *Transfer* tool provides a method for finding such α. The user has to provide theorems of the forms $R_1\, a\, b$ or $(R_1 \Rightarrow R_2)\, f\, g$ (and their proofs), i.e., instances of a structural transformation, and the tactics `transfer` and `transfer′` will try to automatically infer a sentence α such that $\alpha \longleftrightarrow \beta$ (in the case of `transfer`), or a weaker one such that $\alpha \longrightarrow \beta$ (in the case of `transfer′`).

Recall that the intuitive interpretation of $(R_1 \Rightarrow R_2)\, f\, g$ is 'arguments related by R_1 are mapped to values related by R_2 by f and g. Thus, the first step of the transfer method is to search for a theorem of the structural transformation with the shape $(R_1 \Rightarrow \texttt{eq})\, p\, q$ in the case of `transfer` and $(R_1 \Rightarrow \texttt{imp})\, p\, q$ in the case of `transfer'`, where q is the property wrapping the sentence we want to prove. Finding it would imply that we can replace q by p provided that we can find that their arguments are related by R_1. Thus, the method searches recursively for rules in the structural transformation to prove this. The algorithm is analogous to type inference. It is based on the following derivation rules:

$$\frac{\mathscr{A}^*_{\mathscr{C}} \vdash (R_1 \Rightarrow R_2)\, f\, g \qquad \mathscr{A}^*_{\mathscr{C}} \vdash R_1\, x\, y}{\mathscr{A}^*_{\mathscr{C}} \vdash R_2\, (f\, x)\, (g\, y)}\ \text{elim}$$

$$\frac{\mathscr{A}^*_{\mathscr{C}}, R_1\, x\, y \vdash R_2\, (f\, x)\, (g\, y)}{\mathscr{A}^*_{\mathscr{C}} \vdash (R_1 \Rightarrow R_2)\ (\lambda x.\, f\, x)\ (\lambda y.\, g\, y)}\ \text{intro}$$

where $\mathscr{A}^*_{\mathscr{C}}$ represents knowledge about the structural transformation. Practically, the user provides knowledge specific to this transformation (a set of theorems called *transfer rules*), and the algorithm includes in the search other general transfer rules such as $(\texttt{eq} \Rightarrow \texttt{eq} \Rightarrow \texttt{eq})$ `imp` `imp`. For more details of the actual implementation of the algorithm see [6].

5 Mechanising transformations in Isabelle's HOL

In Section 3 we presented some problems in discrete mathematics which involve structural transformation. We have mechanised the transformations by proving the necessary transfer rules. The transfer tool allows us to use the transformations in proofs.

In this section we present a couple of examples from a larger catalogue of the transformations we have mechanised in Isabelle. The transformations we have formalised, as suggested in Figure 1, are the following:

1. **numbers-as-bags-of-primes**, where each natural number is related to the multiset of its prime factorisation.
2. **numbers-as-sets**, where numbers are related to sets by the cardinality function.
3. **sets-as-$\mathbb{B}$-functions**, where sets are seen as boolean-valued functions.
4. **multisets-as-$\mathbb{N}$-functions**, where multisets are seen as natural-valued functions[6].
5. **sets-as-lists**, where sets are related to lists of their elements.
6. **bits-from-integers**, where type `bit` is created as an abstract type from the integers.

[6] This one is actually by construction using `typedef` and the Lifting package, which automatically declares transfer rules from definitions lifted by the user from an old type to the newly declared type.

7. **bits-as-booleans**, where bits are matched to booleans.
8. $\mathbb{Q}$**-automorphisms**, where rational numbers are stretched and contracted, parametric on a factor.
9. **zero-or-some**, where natural 0 is related to bit 0 and positive natural numbers are related to bit 1.
10. **multisets-as-lists**, where multisets are related to lists of their elements.
11. **set-to-multiset**, where the functional representations of multisets and sets are related (this one, we get it for free from the zero-or-some transformation).
12. **naturals-as-integers**, where naturals are matched to integers (this one was built by the developers of the transfer package, not us).
13. **integers-as-rationals**, where integers are matched to rational numbers. Notice that composition of transformations leads to other natural transformations, like the simple relation between sets and multisets.[7]

Every transformation starts with a declaration and proof of *transfer rules*, which are sentences satisfied by the structural transformation.

5.1 Numbers as bags of primes

The relation at the centre of this transformation is $\mathcal{F} : \mathbb{N} \to \mathbb{N}\ \mathtt{multiset} \to \mathbb{B}$, which relates every positive number to the multiset of its prime factors. It is defined as follows: $\mathcal{F}\, n\, M$ holds if and only if

$$(\forall\, x.\ \mathtt{count}\, M\, x > 0 \longrightarrow \mathtt{prime}\, x) \wedge n = \prod_{x \in M} x^{\mathtt{count}\ M\, x}$$

The most basic transfer rules (instances of the structural transformation) are theorems such as $\mathcal{F}\ 6\ \{2, 3\}$, whose proof are trivial calculations. Moreover, from the Unique Prime Factorisation theorem we know that $\mathcal{F}$ is bi-unique. Thus, we know that

$$(\mathcal{F} \Rightarrow\!\!\!\!\Rightarrow \mathcal{F} \Rightarrow\!\!\!\!\Rightarrow \mathtt{eq})\, \mathtt{eq}\, \mathtt{eq}$$

i.e., that equality is preserved by the transformation.

From the fact that every positive number has a factorisation we have

$$((\mathcal{F} \Rightarrow\!\!\!\!\Rightarrow \mathtt{revimp}) \Rightarrow\!\!\!\!\Rightarrow \mathtt{revimp})\ \forall_{>0}\ \forall \qquad ((\mathcal{F} \Rightarrow\!\!\!\!\Rightarrow \mathtt{revimp}) \Rightarrow\!\!\!\!\Rightarrow \mathtt{revimp})\ \exists\ \exists_p$$
$$((\mathcal{F} \Rightarrow\!\!\!\!\Rightarrow \mathtt{eq}) \Rightarrow\!\!\!\!\Rightarrow \mathtt{eq})\ \forall_{>0}\ \forall_p \qquad ((\mathcal{F} \Rightarrow\!\!\!\!\Rightarrow \mathtt{eq}) \Rightarrow\!\!\!\!\Rightarrow \mathtt{eq})\ \exists_{>0}\ \exists_p$$

where `revimp` is reverse implication, $\forall_p$ is the bounded quantifier representing 'for every multiset where all its elements are primes' and $\forall_{>0}$ is the bounded quantifier representing 'for every positive number', and similarly for $\exists_p$ and $\exists_{>0}$. The mechanised proofs of these sentences follow relatively straightforward from the Unique Prime Factorisation theorem which is already part of Isabelle's library of number theory.

[7] The mechanisation of these transformations have been submitted to the Archive of Formal Proofs, along with some examples of their use.

Furthermore, we proved the following correspondences of structure:

$$(\mathcal{F} \Rightarrow \mathcal{F} \Rightarrow \mathcal{F}) * \uplus \qquad (\mathcal{F} \Rightarrow \mathcal{F} \Rightarrow \mathcal{F})\,\texttt{lcm}\,\cup$$
$$(\mathcal{F} \Rightarrow \mathcal{F} \Rightarrow \mathcal{F})\,\texttt{gcd}\,\cap \qquad (\mathcal{F} \Rightarrow \mathcal{F} \Rightarrow \texttt{eq})\,\texttt{div}\,\subseteq$$
$$(\mathcal{F} \Rightarrow \texttt{eq} \Rightarrow \mathcal{F})\,\texttt{exp}\,\texttt{smult} \qquad (\mathcal{F} \Rightarrow \texttt{eq})\,\texttt{prime}\,\texttt{sing}$$

Application in proofs

We formalised the proof of problem 1:

> *Let n be a positive integer. Assume that, for every prime p, if p divides n then p^2 also divides it. Prove that n is the product of a square and a cube.*

Formally, we state this as

$$\forall\, n > 0.\ (\forall\, p > 0.\ \texttt{prime}\ p \wedge p\ \texttt{div}\ n \longrightarrow p^2\ \texttt{div}\ n) \longrightarrow (\exists\, a > 0.\ \exists\, b > 0.\ a^2 * b^3 = n)$$

Notice that the quantifiers of n p, a and b are bounded (greater than 0). This is not necessary (e.g., p is prime, so it is redundant to say that it is positive), but it is convenient for the proof. If we want a proof for the unbounded version (which is also a theorem) we can divide in cases, when $n = 0$ and when $n > 0$. The case for $n = 0$ is trivial because then $a = 0$ and $b = 0$ are solutions. Thus, we prove prove directly the case for $n > 0$.

When we apply the transfer method to the sentence we get the following sentence about multisets:

$$\forall_p\, n.\ (\forall_p\, p.\ \texttt{sing}\ p \wedge p \subseteq n \longrightarrow 2 \cdot p \subseteq n) \longrightarrow (\exists_p\, a.\ \exists_p\, b.\ 2 \cdot a + 3 \cdot b = n)$$

where $\forall_p$ is the universal quantifier bounded to prime numbers, and operator $\cdot$ represents the symmetric version of the multiplication previously referred to as `smult` (we present it as we do for reading ease).

The premise $(\forall_p\, p.\ \texttt{sing}\ p \wedge p \subseteq n \longrightarrow 2 \cdot p \subseteq n)$ is easily proved to be equivalent to $\forall\, q.\ \texttt{count}\, n\, q \neq 1$. Then it is sufficient to show

$$\forall_p\, n.\ (\forall\, q.\ \texttt{count}\, n\, q \neq 1) \longrightarrow (\exists_p\, a.\ \exists_p\, b.\ 2 \cdot a + 3 \cdot b = n)$$

With a bit of human interaction, this can further be reduced to proving that, for every element of n, its multiplicity n_i (which the premise says is different from 1) can be written as $2a_i + 3b_i$, or formally:

$$\forall\, n_i : \mathbb{N}.\ n_i \neq 1 \longrightarrow \exists\, a_i.\ \exists\, b_i.\ 2a_i + 3b_i = n_i$$

This problem can actually be solved in a decidable part of number theory (Presburger arithmetic), for which there is a method implemented in Isabelle.

5.2 Numbers as sets

At the centre of this transformation is the relation $\mathcal{C}$ where $\mathcal{C}\,A\,n$ holds if and only if $\texttt{finite}\,A \wedge \texttt{card}\,A = n$.

We first prove trivial cardinality properties like $\mathcal{C}\,\{1 \cdots n\}\,n$, which allows us to consider standard representatives of numbers.

This relation is right-total but not left total, so we have the following two rules:

$$((\mathcal{C} \Rightarrow \texttt{imp}) \Rightarrow \texttt{imp})\ \forall\ \forall \qquad\qquad ((\mathcal{C} \Rightarrow \texttt{eq}) \Rightarrow \texttt{eq})\ \forall_{\texttt{fin}}\ \forall$$

where $\forall_{\texttt{fin}}$ is the universal quantifier restricted to finite sets. Furthermore, the relation is left-unique but not right-unique, so we have

$$(\mathcal{C} \Rightarrow \mathcal{C} \Rightarrow \texttt{imp})\ \texttt{eq}\ \texttt{eq} \qquad\qquad (\mathcal{C} \Rightarrow \mathcal{C} \Rightarrow \texttt{eq})\ \texttt{eqp}\ \texttt{eq}$$

where `eqp` is the relation of being equipotent, or bijectable.

Then, we have the following rules for the structural correspondence:

$$\begin{aligned}(\mathcal{C} \Rightarrow \mathcal{C})\ \texttt{Pow}\ (\lambda x.\, 2^x)\\ (\mathcal{C} \Rightarrow \texttt{eq} \Rightarrow \mathcal{C})\ \texttt{n-Pow}\ \left(\lambda\, n\, m.\, \tbinom{n}{m}\right)\\ (\mathcal{C} \Rightarrow \mathcal{C} \Rightarrow \texttt{imp})\ \subseteq\ \leq\\ (\mathcal{C} \Rightarrow \mathcal{C} \Rightarrow \mathcal{C} \Rightarrow \texttt{imp})\ \texttt{disjU}\ \texttt{plus}\end{aligned}$$

where $\texttt{n-Pow}\,S\,n$ is the operator that takes the set of subsets of S that have cardinality n. Also, $\texttt{disjU}\ a\ b\ c$ means $\texttt{disjoint}\ a\ b \wedge a \cup b = c$ and `plus` is the predicative form of operator $+$.

We have mechanised combinatorial proofs, like the ones for the problems given in Table 2, of theorems using this transformation.

6 Automated change of representation

We have built a tactic that searches within the space of representations given a set of transformations. Then it tries to reason about each representation. Our goal is for it to embody our vision presented in Section 3. This is work in progress, but we address some simple requirements that we have already implemented and present our observations.

6.1 Transformations as sets of transfer rules

As described in Section 4, we consider a transformation as a set of 'base' relations, and a structural extension of them. Then, *knowing* a transformation means knowing instances where the relations and their extensions (with respect to relators such as $\Rightarrow$) hold. These instances of knowledge are what the Transfer package calls *transfer rules*. They are theorems that the user has to prove and, with enough of them provided, the transfer method will transform the goal to an equivalent, or stronger sentence in another domain.

In the traditional use of the Transfer method, there is a single attribute that encompasses all transfer rules. Given a goal, the Transfer method will try to derive an equivalent or stronger subgoal using all the rules with such attribute, with a simple inference mechanism (described in briefly in Section 4.1 and more detailed in [6]). We have packaged each of the transformations described in Figure 1 as a set of transfer rules. Then, our tactic applies the transfer method one transformation at a time.

Transformation-specific language. Each transformation has a set of definitions that are linked by the transfer package. Some of them are defined only for use of the transformation, like `disjU` and `plus` (the predicative version of disjoint union of sets and addition of natural numbers, respectively), or bounded quantifiers. These are necessary for the transfer method to find matches, but theorems will not generally be stated in such terms. Our tactic normalises the language of the goal to suit the specific transformation that is going to be applied.

6.2 Reversing transformations

We have implemented a tool to automatically *reverse* transformations. Let us explain this.

If we want to transform a sentence $p\,a$ to an equivalent one, the Transfer method will search for transfer rules $(R \Rightarrow \texttt{eq})\,q\,p$ and $R\,b\,a$ for some R, q and b. If found, it will transform the sentence to the equivalent one $q\,b$. The fact that the sentences are equivalent means that if we had started with $q\,b$ as a goal, it would have been valid to transform it to $p\,a$. This means that, in theory, the same transfer rules can be used to do inference in one direction or the other, at least when the rules are regarding equivalence. The Transfer method does not do so: if one wants to use a transformation in both directions one has to define two distinct transformations, i.e., two distinct sets of transfer rules (in our example above one needs transfer rules $(R' \Rightarrow \texttt{eq})\,p\,q$ and $R'\,a\,b$, where R' is the reverse of R). A transfer rule always has a 'reverse' version (although only equivalent ones retain full information), so we should be able to get these automatically. We have built a conversion tool that, given a set of transfer rules, will generate all their reverse rules (in a logically valid way, i.e., the reverse version is always equivalent to the original).

Our program uses the following rewrite rules:

$$R\,a\,b \Rightarrow (\texttt{swap}\,R)\,b\,a$$
$$\texttt{swap}\,(R_1 \Rightarrow R_2) \Rightarrow (\texttt{swap}\,R_1 \Rightarrow \texttt{swap}\,R_2)$$

where `swap` simply swaps the place of the arguments of a function. It is easy to see that these rules are valid. Moreover, $\texttt{swap}\,R$ equals R when R is symmetric, which means that in some relations we can drop the `swap` function. Thus, our program drops `swap` from `eq` and turns `swap imp` and `swap revimp` into `revimp` and `imp`, respectively.

By reversing every transformation we can traverse every path in Figure 1 in any direction (which does not mean that every sentence has a transformation to an equivalent one).

6.3 Search between representations

Our tactic searches the space of representations by applying each transformation, then reasoning within the theory where it arrived, and, if there are still open subgoals it will repeat the process iteratively.

Recall that transformations are relational. As such, the process is non-deterministic for each transformation, so there will be many branches per transformation. Apart from being non-deterministic, the transfer method will allow transformations of a sentence where some matches are left open, i.e., in the place of some constant we get a *schematic variable* that the user can instantiate manually, and prove their validity with the new instantiation. This can be handy, but our tactic favours branches with the lowest number of open subgoals, thus favouring complete matches; e.g., matches that will not leave any proof obligations open.

We have also noticed that the order in which the transformations are searched is crucial and have set an ad hoc order that favours the transformations we consider more interesting. Heuristics deserve further work, but that remains as a task for the future.

Discarding false representations. Recall that our transformations do not necessarily yield equivalent sentences when applying the transfer algorithm (unless we restrict it to do so). Actually, the *numbers-as-sets* transformation can only be applied in useful ways if we allow the reduction of the goal to a strictly stronger subgoal (because, e.g., $A \subseteq B$ implies that $|A| \leq |B|$, but not the other way around, meaning that we can prove $|A| \leq |B|$ by showing first $A \subseteq B$, but we cannot prove $A \subseteq B$ by showing $|A| \leq |B|$). This can lead to false subgoals. Thus, our tactic calls the counterexample checker nitpick [1] and discards branches where a counterexample is found for one of its goals.

6.4 Overview

In a single step in the search, our tactic does the following:

1. Normalise to transformation-specific language.
2. Apply transformation.
3. If working with a transformation that generates a stronger subgoal, search for counterexamples and discard if they are found.
4. Apply `auto` tactic to transformed sentence.

The tactic can be applied recursively to search for a transformation to a domain more than one step away. When searching, the obvious stop condition is that the theorem has been proved, although there can be other good reasons to stop in a domain to allow the user to reason interactively.

Each of the 4 steps mentioned can have plenty of branches, so there is search involved. Branches with the least number subgoals are favoured, and the order in which the transformations are applied matters, but there are no clever heuristics involved.

Even though our observations about the trace of the search have led us to the current design and implementation of the tactic, the design is not yet complete and its implementation (although functional) is very much subject to change. There are still open questions regarding what search strategies, *stop* conditions, and reasoning tactics (between transformations) are the best, because these are subject to what evaluation criterion we should use. In Section 8 we discuss why this is problematic and how we are confronting it.

7 Related Work

Although representation is widely recognised as a crucial aspect of reasoning, to our knowledge there has been no attempt to incorporate the *automatic* search of representation into reasoning tools.

Institutions and HETS

The concept of Institution was introduced to as a general notion of logical system [5]. The Heterogeneous Tool Set (HETS) [8] was developed mainly to manage and integrate heterogeneous specifications. Based on the theory of Institutions, it links various logics, including Isabelle's HOL and FOL, and provides a way of translating between them. The uses of HETS have been to bring together various aspects of complex systems where different programming languages and reasoning tools are used for different parts of the system. We do not know of any uses of HETS where heterogeneity is taken advantage of as a means of finding proofs in one representation where other representations fail.

Little Theories and IMPS

"Little Theories" is the notion that reasoning is best done when it is modular [3]. IMPS is a an interactive proof system implemented based on the principles of Little Theories [4]. The modules, or 'little theories' of IMPS are small axiomatic theories connected by *theory interpretation*. Thus, it concerns different levels of abstraction of a theory, and not directly representation of the entities of the theory.

Uses of the Transfer package

The use of the Transfer package has changed how new quotient types and subtypes are defined. This is what the *Lifting* package does [6]. As part of the lifting package, there is a way of automatically transferring definitions from an old type to a new type (e.g., multisets are defined as an abstract type from the type of $\mathbb{N}$-valued functions).

The Lifting package has been the main application of the Transfer package, although the generality of their approach is acknowledged by the developers. Embodying this generality, they have built an Isabelle theory of transference from integers to natural numbers, very much in the spirit of the various transformations we have built ourselves.

8 Evaluation, Future Work and Conclusion

The main contributions presented in this paper are:

- We mechanised various useful transformations observed in proofs of discrete mathematics.
- We have proved example theorems using these transformations.
- We have identified some requirements for search over the space of representations, and implemented both a tool (for reversing transformations) and a tactic fulfilling the requirements.

Our tactic has yet to be evaluated properly. Below we examine some of the difficulties associated with this task.

What makes one proof better than another? There is no definite answer for this question. Simple measures, such as length, are important, but unsatisfactory as a whole. At the very least, we can agree that some proof is better than no proof. Thus, the simplest scenario for evaluation would be that in which our tactic that reasons within many representations finds proofs which cannot be found otherwise. Unfortunately, the current state of automatic theorem provers does not seem to be conducive to this. All the examples in which we have tested our techniques belong to either of the following classes:

1. They are so simple that they can be proved automatically[8] without the need of a transformation.
2. They are too complicated and require an intervention from the user to complete the proof, even after automatically applying a transformation.[9]

Thus, the proof-or-no-proof criterion is not applicable . Then, it is necessary to work on close analysis of interactive proofs with transformations and without them.

A venture for future research is the potential application of this framework for the transformation of geometric problems into algebraic representations, e.g., Gröbner bases [2], where there has been plenty of success in automated reasoning, or into SAT/SMT, which also have been an area of success in automation.[10]

Interestingly, we have an example (Pascal's theorem) that belongs to the class of problems where Isabelle's automatic tactics can find a proof, but where its proof using a transformation deserves attention. It is provable automatically (from the definition of the `choose` operator included in Isabelle's combinatorics library by its developers), but can be transformed using the numbers-as-sets transformations and proved only interactively there. Arguably, a combinatorial proof could be highly valued by mathematicians (or a scientist who analyses

[8] using Isabelle tactics like `auto`

[9] The examples of this second (more interesting) class have been selected from either maths textbooks for undergraduate students, or from training material for contests such as the Mathematical Olympiads.

[10] We thank the anonymous referees of this paper for suggested these possibilities. They remain as future work.

proofs), making this an example where the interactive proof deserves equal, or even more, attention than the automatic proof.

Furthermore, even in the case in which we had automatic proofs using the usual tactics (like Pascal's theorem, mentioned above), we have to consider that these tactics depend on background knowledge (in our case, this amounts to Isabelle's libraries, which have been vastly populated by users). This raises the question: are there ways in which we can measure success independently of the background theories? We think that this is partially achievable by building simpler theories, with some equal level of measurable simplicity, and testing tactics that incorporate representational change there. Even if impractical by itself, this might bring some scientific insight that might lead to better reasoning tactics and theorem provers in the future.

Bibliography

[1] Jasmin Christian Blanchette and Tobias Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. pages 131–146, 2010.

[2] Bruno Buchberger and Franz Winkler. *Gröbner bases and applications*, volume 251. Cambridge University Press, 1998.

[3] William M Farmer, Joshua D Guttman, and F Javier Thayer. Little Theories. In D Kapur, editor, *11th International Conference on Automated Deduction*, pages 567–581, 1992.

[4] William M Farmer, Joshua D Guttman, and F Javier Thayer. {IMPS} : an Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 9(11):213–248, 1993.

[5] Joseph A Goguen and Rod M Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM (JACM)*, 39(1):95–146, 1992.

[6] Brian Huffman and Ondřej Kunčar. Lifting and Transfer: a modular design for quotients in Isabelle/HOL. In *Certified Programs and Proofs*, pages 131–146. Springer, 2013.

[7] Joe Hurd. System description: The Metis proof tactic. *ESHOC*, pages 103–104, 2005.

[8] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, HETS. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 519–522, 2007.

[9] Tobias Nipkow, Lawrence C Paulson, and Makarius Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer-Verlag, 2013.

[10] Lawrence C Paulson and Jasmin Christian Blanchette. Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers. *PAAR@IJACR*, 2010.

[11] Tjark Weber. SMT solvers: New oracles for the HOL theorem prover. *International Journal on Software Tools for Technology Transfer 13.5*, pages 419–429, 2011.